Semiannual Technical Summary

Graphics

31 May 1971

Prepared for the Advanced Research Projects Agency
under Electronic Systems Division Contract F19628-70-C-0230 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Lexington, Massachusetts

Approved for public release; distribution unlimited.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

GRAPHICS

SEMIANNUAL TECHNICAL SUMMARY REPORT
TO THE
ADVANCED RESEARCH PROJECTS AGENCY

1 DECEMBER 1970 – 31 MAY 1971

ISSUED 1 JULY 1971

LEXINGTON MASSACHUSETTS

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Advanced Research Projects Agency of the Department of Defense under Air Force Contract F19628-70-C-0230 (ARPA Order 691).

SUMMARY

The hardware for the Terminal Support Processor (TSP) system is in various stages of design, construction, and checkout. The two Digital Scientific Corporation Meta 4 microprocessors and main memory system have been delivered, but acceptance awaits the installation of revised memory controller and central processor boards. All components for Lincoln Laboratory's LX-1 microprocessor, the third processor in the TSP system, are constructed and LX-1 is currently being checked out. The secondary memory has been delivered to the interface vendor, and the interface has been checked out on a single processor system. Equipment for interfacing the other peripheral devices is being designed and constructed at the Laboratory. A multichannel controller which will simplify the interface design has been constructed and tested, and the device-specific boards are in various stages of construction and test. A serial transmission and time multiplexing scheme for handling keyboards and console indicators has been designed. The design uses a standard TV composite video signal. Two minimal console stations have been built and tested. Error rates appear to be satisfactory for this application.

TSP software effort has concentrated on developing the support facilities which run on TX-2 and handle assembly and compilation for the TSP. An assembler for LIL, the user language in the TSP, has been designed. Code generators for a BCPL compiler for the BCOM machine, the processor for system software in the TSP, are being debugged and indicate good efficiency of compiled code. Diagnostics for the BCOM machine have been written and debugged on the Meta 4 simulator on TX-2 and have been tested satisfactorily in the Meta 4 system.

The communications hardware for connecting the Laboratory's IMP to the ARPA computer network was delivered and has been operating since January 1971. The IBM 360/67 Network Control Program (NCP) following the old official protocol was operating in February. TX-2 has made use of the network facility using interim network software. New NCP's for both machines are under development.

A structure definition facility is being added to the TX-2 BCPL compiler. Its installation in the front end of the TX-2 BCPL compiler will also make the features available to the BCOM BCPL compiler. A description of the structure definition facility is presented in some detail. Other TX-2 activity has included an air traffic control application which makes use of the BCPL graphics capability.

Work has started on a new program directed toward the development of speech understanding systems. A speech data base to support research in this area has been designed. Programs have been written to facilitate the transmission

of processed speech from the Laboratory's Fast Digital Processor to TX-2 and for the display of speech spectrograms and time functions on the direct view storage scopes at TX-2.

Hardware for interfacing a Xerox LDX Receiver terminal to a PDP-8/L computer and the latter to TX-2 has been constructed and checked out. The device will provide a raster scan printer capability as soon as the necessary TX-2 software changes can be implemented. A Hughes LCSC-1 is being connected to TX-2 for evaluation as an alternative to the direct view storage tube for applications requiring brighter images, higher contrast, or black-on-white pictures. The results of a preliminary investigation were encouraging.

Accepted for the Air Force
Joseph R. Waterman, Lt. Col., USAF
Chief, Lincoln Laboratory Project Office

CONTENTS

GLOSSARY

APEX	The TX-2 time-sharing system
BCOM	BCPL Compilation Oriented Machine – the simulated machine for executing TSP software
BCPL	Basic Combined Programming Language – an intermediate-level language for computer programming
CMS	The monitor system most widely used by on-line users of Lincoln Laboratory's IBM 360/67 computer
CP	Control Program – a time-sharing system for the IBM 360/67 computer
FDP	Fast Digital Processor – a Lincoln Laboratory computer designed for waveform processing applications
LIL	Local Interaction Language – a language for use in the TSP system
NCP	Network Control Program
TSP	Terminal Support Processor

GRAPHICS

I. TERMINAL SUPPORT PROCESSOR (TSP) SYSTEM

The TSP system is a small-scale computer system intended to support interactive graphics users of a computer network. The design aims at providing basic interactive graphics services for a number of consoles, each consisting of a keyboard, a tablet, and a pair of storage scopes. The system provides a language called LIL, which a user can utilize to control interactions between his console input and output devices and between the TSP and other computers in the network. The TSP itself consists of three microprocessors sharing 65 k words of 900-nsec core memory arranged in 8 banks of 8 k words each. Previous reports in this series have described the user specifications for LIL (31 May 1970, DDC AD-709187) and the system architecture of the TSP (30 November 1970, DDC AD-716817).

The following sections contain brief reports on the current status of the hardware and software components of the TSP system. Although the reports show substantial delay in the availability of the processors, the impact of this delay on the development of peripheral hardware and TSP software has been substantially reduced by the availability of the SEL test computer and the microcode simulators running on TX-2. Current schedules indicate that all system hardware will be available by the end of the next reporting period.

A. TSP Hardware

1. Processors and Memory

The Meta 4 dual-microprocessor computer system was delivered on schedule in January 1971; however, two main problem areas appeared: the machine speed did not meet specifications, and the memory controllers were not able to handle an eight-bank configuration. The manufacturer is solving these problems by redesigning the memory controller board and some CPU boards. Acceptance of the machine is awaiting arrival and installation of these boards. Delivery had been expected by this time but has been delayed due to a Digital Scientific Corporation (DSC) plant shutdown. DSC is back in operation and delivery is expected next quarter.

All system components for the Laboratory's LX-1 microprocessor, the third processor in the TSP system, are constructed and the system is being checked out using the SEL test computer. The SEL machine simulates both the Meta 4 memory connections and the display generator which LX-1 drives in the TSP system. Checkout is progressing well; the microprogram memory and clock generators are operational and the CPU section is ready to connect and check out. Microcode diagnostic programs have been prepared on the TX-2 assembler.

2. IO Connections

In the course of designing the circuit boards which interface the peripheral equipment to the Meta 4 system, it became apparent that simplification and lower cost would be achieved by a multichannel controller capable of handling four low-to-moderate speed cycle-stealing devices. These include the two IMP channels, the tablet multiplexer, and the channel for loading LX-1's

writeable microprogram memory. The multichannel controller consists of a single board containing a 16-word-by-16-bit integrated circuit memory to store four control words for each of the four devices. It also provides a common facility for address and transfer count incrementing and other "housekeeping" activities for the channels. The remaining logic peculiar to each device is located on additional boards. The multichannel board has been constructed and partially tested in the Meta 4 system.

The following paragraphs discuss the status of each of the IO devices in the TSP system.

a. Secondary Storage

The 1 M-word Digital Development Corporation disk has been delivered to DSC, and the IO controller board designed and constructed by DSC. This unit has been debugged on a single-processor, single-bank memory Meta 4 at DSC and now awaits final checkout on a TSP-like Meta 4 system.

b. IMP

The two IMP connections (input and output) make use of the multichannel controller board. A board containing the IMP interface logic has been completed and is ready to test.

c. LX-1

LX-1 is connected to the Meta 4 system as a pair of IO devices. One device is a channel which loads the LX-1 microprogram memory and is handled by the multichannel controller board. The other device represents LX-1's access to main memory for programs and data. A board to interface LX-1 for both purposes is presently being laid out.

d. Displays

The display and character generators have been available since the previous report in this series. They are awaiting the completion of LX-1 checkout for the next step in system integration.

e. Keyboards and Indicators

A serial transmission and time-multiplexing scheme has been devised for these low data rate devices. One motive behind the design is to reduce the multiplicity of transmission cables often associated with a multiconsole installation. Another motive is to simplify the priority, or scheduling, mechanism at the processor interface. The resultant design is a synchronous system, where the consoles are polled sequentially in a fixed order. The data transmission is in self-clocking serial form. A standard TV composite video signal is used for the data transmission as well as for the controlling of the multiplexing process. Specifically, each field (1/60 sec) of the composite video is divided into odd- and even-numbered lines by counting the line, or horizontal, sync pulses. The odd-numbered lines are assigned to the consoles for input purposes. Assigned to each console will be a few specific line numbers which are considered to be the console's address, and the console can input data to the TSP only during those times. The consoles' addresses are interlaced so that the access possibilities of the consoles are equally distributed. The even-numbered lines are reserved for the TSP for output purposes. There is no restriction on the TSP as to which console it may address at any time. Instead, each output message carries its destination address. All consoles listen, but only the addressed console accepts the message. Thus, the multiplexer works on a simple queuing scheme for inputs but on a broadcasting scheme for outputs.

The messages are sent during the video portion of the composite video signal and are self-clocked using a frequency-doubling code. For transmission of a "zero," a single transition is made during each bit time; for transmission of a "one," two transitions are made. The first transition of any bit is used to reclock. To guard against errors, a simple parity checking bit is provided with each message. A parity error will cause the rejection of that message. To back up this possible loss of a message, a send-and-acknowledge scheme is used. The sending console retains the message and retransmits it at the next opportunity unless a specific acknowledgement is received. This feature permits the TSP program to effectively lock the console keyboard if it so chooses.

Two minimal console stations have been built and connected to our SEL test computer for error rate measurements. Sixteen-bit pseudo-random words are used. The transmission path is a video cable 150 feet long. The send-and-acknowledge scheme is not used. Test results to date indicate 12 errors in 60 M-word transmissions. We think that with the send-and-acknowledge provision the error rate will be more than adequate for use in the final system.

The design of the interface board for the keyboard-indicator multiplexer is complete, and construction will begin when the LX-1 board is complete. The keyboard-indicator multiplexer will appear as a programmed IO device to the Meta 4 system.

f. Tablets

Although the design of the tablet multiplexer logic is firm, actual construction has been deferred until more of the other connections have been built and tested. The tablet data transfer will be handled by the multichannel controller board.

g. Miscellaneous IO Devices

More detailed information on Meta 4 timing is required before the final design can be carried out of an IO device to capture information about memory parity errors and protection violations occurring in the Meta 4 IO system. This device will also facilitate program-controlled interprocessor interrupts. The necessary information is expected to be available when the new Meta 4 memory controller and CPU boards are delivered.

B. TSP Software

Since the Meta 4 system has not been available for program checkout, TSP software effort has concentrated on developing the support facilities which run on TX-2 and handle assembly and compilation for the TSP. Work on programs for TSP execution has been limited to firmware (microcode) which can be debugged on the TX-2 simulator. The Meta 4 microprocessors in the TSP system will execute microcode to simulate two distinct 16-bit processors. One, called the BCOM machine, will execute TSP system programs; the other, called the LIL machine, will execute user programs. In the following paragraphs, the term BCOM and LIL will be used to refer both to the simulated machines and to the assembly languages for those machines.

1. Meta 4 Simulator

The Meta 4 computer has an operator's panel composed of 7 function switches, 2 rotary switches (with 5 and 7 positions, respectively), 16 data entry switches, and 36 associated indicator lights. The operator's panel is an input-output device on the Meta 4. Communication is handled via the two Meta 4 IO instructions. The operator's panel has been implemented in

the Meta 4 simulator on TX-2 making use of the set of 36 input switches available on three of the TX-2 consoles. Certain of these switches are assigned to represent the 16 data switches, the function switches, and the various positions of the rotary switches. The running of BCOM programs under the simulator is now controlled from the simulated panel. In its implementation of the two Meta 4 IO commands, the simulator attempts to ascertain that the necessary programming and IO timing requirements have been met. The simulator catches both errors which would cause the real Meta 4 to hang up, and errors which would be accepted by the Meta 4 but would produce an incorrect result.

A representation of a Meta 4 operator's panel is displayed on the console's storage scope. It shows the actual positions of the various switches and conditions of the 36 indicator lights. During the running of a BCOM program, this display is updated as needed so that it closely simulates a real operator's panel.

2. BCOM Diagnostics

A set of diagnostic routines has been written to be run by BCOM on a Meta 4. These routines test all the variations of each BCOM instruction which have been implemented in ROM code. They test only that the BCOM instruction set works; they do not check out the Meta 4 in general. In addition, a worst-case memory test program for Meta 4 main memory has been written in BCOM assembly language. Several bugs in the BCOM firmware were discovered and corrected in the course of debugging the diagnostics on the simulator. A brief test of the BCOM diagnostics on the Meta 4 system uncovered no new bugs in the BCOM firmware and suggested that the simulator was reasonably sound.

3. LIL

The final LIL instruction set has been specified, and it varies only slightly from that described in the 31 May 1970 Semiannual Report. The file structure for program and data segments as well as the binary representation of the operation instructions have been specified. The microcode for the operation instruction set has been written and partially debugged using the simulator. Debugging has been slowed by the lack of an assembler for the LIL language, and test programs have had to be coded in octal. Major debugging is being deferred until the assembler is available.

4. LILA

LILA (the LIL assembler) has been designed and is being written in BCPL, which should facilitate its transfer to our IBM 360/67 or any other machine with a BCPL compiler. Wherever possible, use of machine-dependent features of BCPL will be avoided. Any functions in LILA which are of necessity machine-dependent will be isolated in separate routines.

5. BCPL Compiler for BCOM

The TX-2 BCPL compiler is being modified to generate a BCPL compiler for the BCOM machine which will run on TX-2. The new code generators have been written and are partially debugged. A new front end which will provide structure definition facilities has been designed but not yet coded. (See Sec. III-A of this report for a discussion of these structure facilities.)

The code generators take an intermediate language produced by the front end of the BCPL compiler (OCODE) and generate the corresponding assembly code. The intermediate language is designed to be run on a stack machine, and the code generators attend to the details of allocating space from a core area which contains the stack. The code generators recognize that often there is no need to store values which have been pushed into the stack and so a "false" stack is maintained during code generation to remember information such as "the A-register contains the element which is two down from the top of the stack." Furthermore, the fake stack is used to postpone performing such operations as plus and RV (indirect) until a good choice for the register to be used is known.

Running the BCOM BCPL compiler on some parts of itself has allowed us to check the accuracy of the code generators. It has also allowed us to compare the number of instructions which are compiled for a given BCPL program on BCOM, and the number which are required on TX-2. It is interesting to note that, for several programs, these numbers have been approximately the same, despite the fact that TX-2 words are 36 bits long while BCOM words are only 16.

II. ARPA NETWORK

Work is currently under way on interfacing three host computers to the ARPA computer network. The Laboratory's IBM 360/67 is being connected as a facility capable of providing time-sharing services to remote users. Connections for TX-2 and the TSP systems are being developed to explore interactive graphics in a network environment. This section reports the status of work on the network hardware connections and the supporting software for the 360/67 and TX-2. In addition, some work on a general network study is also reported.

A. Network Hardware

Shortly after the IMP was installed in July 1970, we began a series of extensive tests of the interface between the IMP and the TX-2. These tests revealed erratic data errors which occurred, in bursts, at an average rate of about 1 bit per million. The culprit, a faulty TX-2 package, was finally located after about five months. Another half-dozen hardware failures, some in the TX-2 and some in the IMP, were also uncovered during this time. Only a moderate amount of hardware testing has been done since the cause of the erratic failure was found.

The IMP machine was moved to its permanent location in Building J (IBM 360/67 site) in December 1970. A shakedown period of testing the long cables and Distant Host Interface between the IMP and TX-2 produced some problems which were ironed out. In January, it became possible to echo random messages between TX-2 and the IMP with no errors.

The telephone terminal equipment (data sets and modems) and circuits for connection to ARPA Network nodes in Cleveland (Case-Western Reserve University) and Cambridge (M.I.T.) were installed and checked out. Messages were sent between our IMP and IMP's at other nodes in the Network.

The 360/IMP Interface arrived from the vendor on 1 February 1971 and was installed adjacent to the IMP. The Interface was cabled up to the 360 Multiplexer Channel and the IMP Local Host Interface. Basic hardware functions were performed satisfactorily and network operation was achieved on 1 March. Minor difficulties with some features were observed during the ensuing period and were corrected when sufficient documentation became available.

B. 360/67 Network Software

A Network Control Program (NCP) satisfying most of the protocol requirements of network Document 1 was operational in February. A revised NCP satisfying the new protocol requirements of Network Working Group/Request for Comments 107 is currently under development. This section discusses some implementation features of the current NCP which will be carried over into the revised version.

CP-67, the time-sharing system in use on the IBM 360/67 at Lincoln Laboratory, provides virtual System 360 machines for on-line users. Most on-line users utilize CMS, an operating system which runs in the virtual machine, to provide system services to their programs. The NCP, which has been developed for the 360/67, runs in a virtual machine which has direct access to the Interface Message Processor (IMP), and hence to the network, and communicates with other virtual machines over a pair of virtual devices, one for sending and one for receiving. The virtual device used for sending can be directed to a particular virtual machine in the same way that user virtual machines direct transmission to the NCP virtual machine. Transmissions to the NCP are identified as to the sending user and thus are appropriately processed by the NCP.

Implementation of the NCP in a virtual machine rather than as part of the core resident system facilitated its development since it could operate independently of other virtual machines and, hence, not impact the reliability of the operating system. The NCP is written as a user program which calls on CMS system facilities for program initialization, including program reinitialization in the event of a catastrophic error, for disk file services, and for virtual-machine-to-virtual-machine communications. This latter facility utilizes the existing mechanism for converting spooled punched output of one virtual machine to spooled reader input of another virtual machine.

By operating the NCP in a virtual machine, core memory is not dedicated to support the network operation; system resources are only required when network activity occurs. In addition, operating the NCP in a virtual machine is particularly convenient since the protocol over which NCP's communicate is being changed and will continue to evolve. The added overhead to the operation of the NCP caused by these techniques does not preclude useful network operation and experimentation. A number of improvements can be implemented to reduce this overhead by providing special virtual machine facilities and by giving the NCP special priorities if the use of the network is limited by the performance of the NCP.

The communication protocol between a user virtual machine and the NCP resembles the NCP-to-NCP protocol. Data cannot be transmitted until a connection is initiated by both the sender and the receiver and the NCP-to-NCP requirements for flow control are met. The NCP specifies the number of bits it is prepared to buffer for a receive connection and stores messages received from a foreign NCP until the user makes a request for a message. If no message has been received when a user makes a receive request, this fact is returned to the user. The NCP will not accept a message from a user for transmission if the receiving NCP has not provided adequate buffering. In this way, the requirements for flow control between NCP and NCP are passed to the user. The NCP also notifies the user of any asynchronous change in the state of a connection so that he can take appropriate action. For example, the user is notified (1) when a connection he has initiated has been completed by the foreign host, (2) when a message is received and there had not been any previous messages buffered, and (3) when a connection is closed by the foreign host. In addition, the user is notified when a network interrupt has been received for the connection.

A set of low-level user routines (NET) have been provided to communicate with the NCP. These routines follow the protocol defined for network communications through the NCP. For each user request to the NCP, a reply from the NCP is sent to the user virtual machine indicating either successful or unsuccessful completion of the request and the current status of the connection. These routines handle the stacking of asynchronous interrupts which are unstacked through a call to the NET routines.

The LOGGER, which has been developed for the 360/67 system, runs in its own virtual machine. It controls a number of virtual terminals which provide access to the CP-67 system in a manner similar to physical terminals but, instead of a hardware terminal, CP is interfaced with a specially designed virtual IO device which is driven by software. Thus, information normally typed on a terminal by CP and information normally keyed into a terminal is transferred between the LOGGER and CP through one of these virtual devices. When the LOGGER is initialized, it is prepared to communicate with the NCP over the socket defined for LOGGER operation under the Initial Connection Protocol (ICP). Communication under the ICP will result in a new pair of sockets being defined through which the LOGGER will provide CP services to the network. Once this pair of sockets is defined, the LOGGER acts as a SERVER interfacing a virtual terminal with a remote process through the NCP.

During the initial log-in sequence, the LOGGER analyzes received network messages for "userid" and "password" and compares these with entries in a special file. If the userid does not appear in the file, an invalid userid is passed to CP. If the password matches the network password corresponding to the valid network userid specified in the file, the system password is sent to CP. Since network passwords and valid system passwords do not have to be identical, access to the CP system from the network is specially controlled and may be restricted to a subset of authorized system users. Though the operation of the LOGGER/SERVER as a separate virtual machine introduces another level of dispatching overhead, this does not appear to be significant and does not limit its use.

Another program (TELNET) has been written which can be run by any CMS time-sharing user to communicate over the network with a remote LOGGER/SERVER. This program makes use of macros provided to simplify use of the low-level network user routines to communicate with the NCP. TELNET has been used to access the LOGGER at UCLA and to use the UCLA time-sharing system. A UCLA TELNET program was then used to create a new virtual machine on the Lincoln Laboratory 360/67 which accesses CP through our LOGGER/SERVER.

C. TX-2 Network Software

A proposed experiment (see the Semiannual Technical Summary dated 30 November 1970) to gain experience with alternate protocols has not been carried out, and has been postponed indefinitely. The effort required to produce an NCP to satisfy current known requirements is sufficiently great that manpower cannot be made available to work on alternatives for comparison. Current TX-2 network effort is concentrated on design of an NCP which meets the protocol requirements plus the additional requirement of efficiency for use with the TSP system. The following two sections present some observations on experience with an interim network capability, and a discussion of some aspects of network software being developed to satisfy longer-term requirements.

1. TX-2 Interim Network Software

The APEX time-sharing system has been modified to allow a user program to communicate with the network. Any user is allowed to capture all incoming messages and to send arbitrary messages over the network. This approach is reasonably acceptable on an interim basis, since there is not yet sufficient demand for the network to require simultaneous access by several users.

A user program has been written to send arbitrary binary messages, typed by the user in octal, to the network. Messages received from the network are displayed, in octal, on the storage scope. This program, while extremely inconvenient to use, comes in handy when it is necessary to know and control exactly what is transmitted through the network.

A second user program has been written to allow a TX-2 user to log in to the 360/67 at Lincoln over the network. A one-letter command to this program causes it to generate the sequence of messages necessary to "dial up" the 360. Once the user has contacted the 360, he may direct his typing either to it or to the TX-2. Typed output from the 360 is displayed on the storage scope. Another one-letter command to this program causes it to set up a second connection to the 360, and to store the arbitrary binary data which arrive over this connection in a file.

This program was designed to be as simple as possible. It handles only a single user and a specific foreign host. Nevertheless, a fairly complicated program is needed to sort out the variety of messages which arrive from the network and the user, and to generate the appropriate messages for the network. No attempt has been made to handle or even report most of the error conditions which can occur, such as unexpected or missing messages from the network. It seems that a good deal of thought would have to go in to developing automatic means of recovering from these conditions.

The program has been used successfully to log into the 360 and transmit data for the speech project to the TX-2. Operation has not been reliable. Occasionally one computer or the other simply fails to respond. Because of the lack of error handling, it is impossible to pinpoint the difficulty. In such cases, a programmer who is familiar with the network can use the program which sends arbitrary binary messages to the network in an attempt to restore communications. We clearly do not yet have a facility which is satisfactory for the naive user.

2. TX-2 Standard Network Software

To satisfy the official protocol requirements, the TX-2 NCP is being implemented as an integral part of the APEX system because fairly heavy network traffic is expected between TX-2 and the TSP, and because the APEX system lacks the facilities for maintaining a privileged program outside the system.

Several goals have motivated aspects of the design not specified by protocol requirements. First, performance should approach the hardware limits. Second, restrictions on the user program's options should be kept to a minimum. Third, the NCP should not have to allocate system buffer space for arbitrary periods of time.

The NCP is designed to share message buffers with the user. This feature allows the messages to be sent and received without an intermediate copying stage by the NCP. It also prevents tying up system buffer space for messages which the user may not accept for an arbitrary period, and permits inactive messages to move to secondary storage by the use of existing mechanisms. Initially, all buffers will remain frozen in core; later, algorithms for releasing buffers

temporarily inactive will be investigated. The shared buffers are circular, with messages linked together. The size of these shared buffers must, of course, be reflected in the flow-control algorithms.

Rather than attempt to have the NCP manage the user scheduling and flow-control decisions, we decided that the NCP should recognize only simple conditions (e.g., requests to establish or close connections) and should activate a low-overhead user routine to make the decisions. The user will be allowed to specify which conditions he wishes to be notified about, and other conditions will be ignored or will cause interrupts similar to those caused by program errors such as attempts to execute an undefined instruction.

We have elected to dispense with the usual "listen" command, and instead allow the user to request an interrupt on an attempt to establish a connection, and allow the user to determine whether the connection should be accepted. We have further specifically allowed such requests to remain pending while a different connection is open. The NCP will retain the order in which the requests came in, but the user may, of course, choose any order in handling the requests.

The user of a receive socket will be allowed to use the network allocation mechanism for flow control. For example, it is expected that receive sockets providing information for a slow local output device such as a typewriter would wish to limit the bit allocation so that the network could not get more than a few seconds ahead of the output device. Such usage can avoid the necessity for implementing complex "stop typing" or "break in for system message" functions.

Software to permit use of existing TX-2 programs from the network is being developed as a set of user programs which will automatically intercept typeouts and convert them into outgoing network messages, and similarly will direct incoming "typed" commands into text to be placed in a keyboard buffer for the recipient command translation. By developing these programs as interrupt-driven user routines rather than system routines, greater flexibility can be achieved and, at the same time, the details of protocol, buffer size, etc. can effectively be hidden from the user. Current TX-2 user programs which make use only of keyboard typewriter IO should not require modification in order to be used from the network.

Remaining problems include the design of reasonable and convenient translation schemes between the TX-2 character set and the ASCII set. Previous attempts at such translations were not satisfactory from a human factors standpoint.

D. Network Studies

The ARPA Network, as it has been developed, allows a user to have direct access to a remote computational facility on the same basis as a local user of that facility. There is a clear economic advantage to this type of resource sharing; sites with an excess computing power are able to sell the surplus to remote users. Other sites may choose to purchase time elsewhere, rather than to keep under-utilized machines.

Apart from the advantages of remote access, the present network design offers almost no difference in the style of computation possible. There is nothing that can be done remotely that can not presently be done locally. The potential for new modes of operation has, to date, been largely unexplored.

We are participating in a network study group which has been organized with the aims of discovering the potential offered by networks for new styles of computing, and characterizing the problems that must be solved before this potential can be realized. The advantages currently being explored include:

(1) Partitioning problems among nodes best suited for each component.
(2) Increasing throughput by concurrently using many processors.
(3) Controlling access to distributed data bases.
(4) Integrating pre-existing service programs.
(5) Using redundant processing and resource scattering to increase reliability.

We expect that a critical study of these new application areas will allow us to better characterize the types of demands that will be made upon future network facilities. Such advanced forecasts should lead to more stable and cost-effective system implementations. Current activity has been directed toward explicating the demands of user processes on the local NCP, with the idea that such discussion will clarify the goals being pursued and will provide a benchmark for gauging various NCP implementation strategies.

III. LANGUAGES AND APPLICATIONS

A. Structure Facility for BCPL

1. Introduction

An important problem in programming concerns accessing and changing subfields of structured data. Here the term "structured data" refers to any collection of data – that is, of bits – which has some structure meaningful to one or more programs. As a simple example, any compiler that compiles to machine code (as opposed to assembly code) is concerned with the instruction format of the object computer. Specifically, consider the TX-2 whose 36-bit instruction word is divided like this:

H	1	Hold Bit
CF	5	Configuration
OP	6	Operation Code
J	6	Index Register Specification
D	1	Indirect (defer) Bit
AD	17	Address

Here, for each field we give a one- or two-character name, the width in bits, and its function (which is irrelevant to the present discussion). Now consider a compiler written to compile code for this machine. If JJ is a variable containing the index register desired, the command

W = (W ∧ #777700777777) ∨ ((JJ ∧ #77) lshift 18)

might be used to set the index part of W. (Here '∧' is the logical and operator, '∨' is the logical or operator, and the '#' signifies an octal constant.) If X is a pointer to an instruction, then the command

rv X = (rv X ∧ #777700777777) ∨((JJ ∧ #77) lshift 18)

might be used instead. It would clearly be desirable to be able to program this operation in a more transparent manner. It is this sort of problem that the structure definition facility about to be proposed helps to alleviate.

Let us continue with the above example. In the syntax about to be described, the instruction format given above might be described by the following structure declaration:

```
structure Instruction

{   H     bit   1   // hold bit
    CF    bit   5   // configuration
    OP    bit   6   // operation code
    J     bit   6   // index register specification
    D     bit   1   // indirect address
    AD    bit  17   // address
}
```

This declaration defines the identifier "Instruction" as being associated with a structure, the structure being composed of fields of bits as shown. The dot is used to indicate substructure, so that

Instruction·J

refers to the J-part (that is, the index part) of an instruction. We can then refer to the index part of the word pointed to by X as

X ⊃ Instruction·J

The mark '⊃', which may be read 'right lump', has been selected because of its resemblance to a pointer. It indicates that we are concerned with a subfield of a word pointed to. If, instead, W were the actual word in question instead of being a pointer to that word, we might write

W ⊂ Instruction·J

The '⊂' may be read as 'left lump'; it has no particular mnemonic significance but is clearly the opposite of a ⊃. The statements given earlier might then be written as

W ⊂ Instruction·J = JJ
X ⊃ Instruction·J = JJ

respectively. These forms are clearly more transparent. Similarly, the 'hold' bit of the instruction pointed to by P could be set to one by executing

P ⊃ Instruction·H = 1

A structure facility is to be added to BCPL to permit convenient access to and changing of subfields of words. An important advantage of doing this is that the description of data bases can be separated (in separate 'get' files fetched by the compiler) from the code that massages them. The idea is to be able to specify a 'shape' of a data item—the way it looks in core. The shape of a datum is, in general, distinct from its location.

2. Syntax

Three new constructs are to be added to the language—the <structure declaration>, the <structure reference>, and the 'size' expression. The first may be used wherever a declaration may be and serves to declare that a particular identifier references a particular structure, or shape. The second may be used wherever a primary may be used and serves to access a structured item. The last may be used to get, as a constant expression, the size in words of a structure.

BNF syntax follows, using the usual notation. Names of syntactic classes are enclosed in angle brackets, the vertical bar '|' is the meta-linguistic or, and '::=' means 'is defined to be'. All other characters stand for themselves.

Syntax for Structure Declarations

```
<Structure Declaration> ::=
      structure {<SD-List>}

<SD-List> ::=
      <SD-Item>  |  <SD-Item>; <SD-List>

<SD-Item> ::=
      <SD-Term>  |  <SD-Term> ≡ <SD-Item>

<SD-Term> ::=
            <Name> <Replicator> <Declarator> <Size>
      |  fill byte  |  fill word
      |  <Name> <Replicator> (<SD-List>)
      |  (<SD-List>)

<Replicator> ::=
          ∩ <Constant Expression>
      |  ∩ <Constant Expression> ∩ <Constant Expression>
      |    <Empty>

<Declarator> ::=
      bit  |  byte  |  word  |  bitn  |  byten  |  char

<Size> ::=
      <Constant Expression>  |  <Empty>
```

Syntax for Structure References

```
<Structure Reference> ::=
            <Expression> ⊃ <SRI>
      |     <Expression> ⊂ <SRI>

<SRI> ::=
      <SRC>  |  <SRC>  ·  <SRI>

<SRC> ::=
      <Name>  |  <Name>  ∩  <Expression>
```

In addition, add to the definition of <Expression> the possibility

```
|  size <Constant Expression>
```

The syntactic categories left undefined in this syntax are

<Expression>	any expression
<Constant Expression>	any expression whose value can be deduced at compile time
<Name>	any identifier

3. Semantics

A <Structure Declaration> is clearly a list of <SD-Item>s, separated by semicolons. (The semicolons missing from the examples shown throughout this section would be inserted automatically by the compiler.) Each <SD-Item> is one or more <SD-Term>s, separated by '≡'s. Ignoring this possibility for the moment, assume an <SD-Item> to be an <SD-Term>. The interest comes in an <SD-Term>, of which there are four flavors. As an example of the first, consider

```
X    bit    5
```

This specifies a field named 'X' which is 5 bits wide. The <SD-Term>

```
Y∩3    bit    7
```

specifies three replications of field 'Y', each replication being 7 bits wide. The 'up lump' indicates a structure subscripting operator, used in both declarations and reference. The three

instances of Y would be referred to as Y∩1, Y∩2, and Y∩3. Note that the first has 'subscript' 1, as opposed to the usual BCPL subscription convention in which the first has 'subscript' zero. The difference between structure subscripting and regular BCPL subscripting is emphasized by using different characters. Now consider

Z∩0∩2 bit 7

This uses as much space as the previous example, but the fields are to be referred to as Z∩0, Z∩1, and Z∩2. We see then that if the replicator is absent, it is taken as one. If one value is given (as for Y above), it is taken as the upper limit with the lower limit taken as one. If two fields are given, they are taken as the lower and upper limits, respectively.

Consider now the classes <Declarator> and <Size>. The keywords 'bit', 'byte', and 'word' are self-explanatory, although the number of bits in a byte and of bytes in a word are, of course, implementation dependent. In most implementations, 'char' would be the same as 'byte' but they may be different. (Even if they are identical, the programmer may find it convenient to think of some items as char and some as byte.) The declarators 'bitn' and 'byten' refer to numeric fields. When referenced, they are taken as signed quantities and treated as appropriate in the implementation. For example, in a computer using two's-complement arithmetic, the leftmost bit of the field would be copied into all bits of the word to the left of the field when fetched.

The <SD-term>s 'fill byte' and 'fill word' represent fields wide enough to go to the next byte boundary or word boundary, respectively. No name is associated with these, as they are used only to insure that the next field starts on an appropriate boundary. Since a subfield may not extend over a word boundary, this is frequently necessary. Use of fill frequently permits a given declaration to be used on different computers with different word lengths.

The <Size> specifies the width of the field, in units of the <Declarator>. If missing, it is taken as one. Thus, byte 3 refers to a field 3 bytes wide.

All that remains to explain is the '≡'. Two <SD-Term>s separated by a '≡' are to occupy the same storage. Consider, for example, the declaration

structure {A (B byte 2; C byte 2 ≡ CN byten 2)}

The reference X⊂A·B refers to the left half of X, and X⊂A·C refers to X's right half. (This example assumes 4 bytes per word.) However, X⊂A·CN interprets X's right half as a numeric quantity, so that it would be accessed with sign extension. That is, C and CN refer to the same part of the structure. Consider another example, in which we assume four 9-bit bytes per word:

structure {A (B byte 2; C∩6 bit 3 ≡ D∩3 bit 6)}

Here, the right half of the word is to be regarded as either three 6-bit fields or six 3-bit fields.

4. Examples

Following are some examples of <Structure Declaration>s, with comments on their effect. The examples assume a 36-bit word with four 9-bit bytes per word, as on the TX-2.

A string in BCPL on the TX-2 is stored four characters per word, with the length (in characters) stored in the first byte position. (The BCPL convention is that byte positions are counted from left to right.) Then a declaration for such a structure is

structure {String (N byte; C∩511 byte)}

With this declaration, the length of string X may be referred to as X⊃String·N, and the fourth

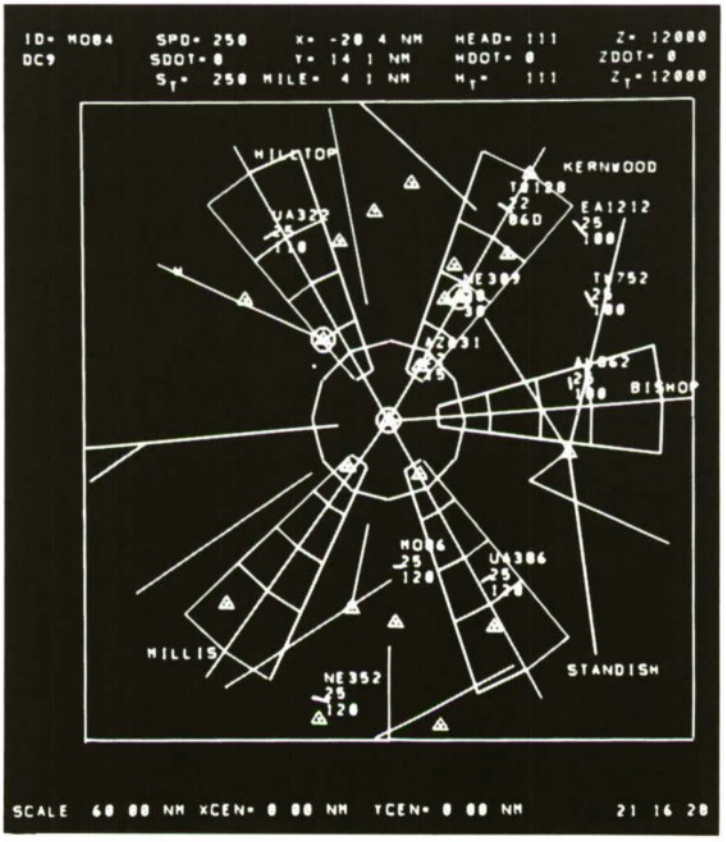


Fig. 1. Typical display produced by air traffic simulatian pragram. Aircraft are represented as radar slashes 4° wide, with alphanumeric infarmatian ta identify flight number, speed, and altitude. At tap af screen is detailed infarmatian regarding ane specific aircraft.

character of X as X⊃ String·C∩4. The number 511 in the declaration comes from the fact that the maximum string length must be storable in 9 bits. The length of a string in words is given by the expression

```
size String
```

This expression has value 128 (given the above declaration of string on the TX-2) and is capable of being evaluated at compile time.

Sometimes it is convenient to store strings one character per word. A useful format is to put the length in the zeroth word of a vector and the characters in successive words. Routines for unpacking and packing strings are then like this:

```
let UnpaekString (S , V) be // Unpack string S into vector V
{     V|0 = S⊃ String·N // The Length of the String
      for K = 1 to V|0 do V|K = S⊃ String·C∩K
}

and PackString (V , S) be // Pack vector V into string S
{     S⊃ String·N = V|0
      for K = 1 to V|0 do S⊃ String·C∩K = V|K
}
```

Note the rather pleasant symmetry between these two routines.

Here is a routine that reverses the bits of a word:

```
let Reverse (X) = valof
{     structure {B∩36 bit 1} // 36 1-bit items
      let T = 0
      for N = 1 to 36 do  T⊂ B∩N = X⊂ B∩(37-N)
      resultis T
}
```

The value of the function is the reverse of its input.

B. Air Traffie Control Applieation

The application program discussed in this scction was supported by another Laboratory program. However, it made use of programming tools and techniques developed under the Graphics program, and is prescnted here as an illustration of the effectivcness of these tools in a particular application situation.

A simulation of air traffic in a terminal area has recently been programmed on TX-2 in BCPL. The scope (Fig. 1) is organized in a manner similar to the display seen by air traffic controllers, and the operator may issue commands to change parameters such as heading, altitude, and speed. A table of characteristics of various aircraft is used to generate realistic accelerations, rates of turn, etc. A script of arrivals and departures may be provided to the program or generated by the operator as he goes. A facility for replaying an entire session has been provided as well as a "snapshot" facility for saving the instantaneous state of the system. The intent is to evaluate several different proposed air corridors for the area around Logan Airport by having controllers from the Logan tower play through identical scripts of arrivals and departures with different maps and observing the time and mileage for each aircraft in the system.

The program required approximately four man-weeks to write, debug, and adjust until it looked realistic. With each step in the simulation representing 4 sec of simulated time, we can handle 45 aircraft simultaneously at real-time rates. Note that TX-2 has no floating-point hardware nor any hardware aids for scaling and windowing. Even so, it is estimated that, if necessary, this number could be increased by a factor of two or three with a relatively small amount of additional work.

IV. SPEECH

In anticipation of a reorientation of the Graphics program toward work on speech understanding systems, a small effort directed toward the development of a speech data base has been started. The data base is expected to grow and eventually to contain a substantial quantity of labeled utterances. In addition to the speech waveform itself, the data base would also contain "front-end" processing, such as the computation of spectra, pitch, and formants, and of higher-level processing, such as phoneme recognition and syntactic and semantic analyses. Front-end processing is planned to be carried out on the Laboratory's Fast Digital Processor (FDP), a high-speed processor designed particularly for waveform processing applications. Higher-level processing and data base management activities are expected to be carried out on TX-2. Current activities are directed toward the design of the data base and the development of programs to support the processing and labeling of speech samples. The general structure of the speech data base has been laid out, but many details have still to be dealt with, and a discussion of the data base organization is best deferred until a later report. The remainder of this section will be concerned with a report on some of the support programs which have reached operational status.

Our plan calls for a direct connection between TX-2 and the FDP for the rapid transmission of processed speech data, but such a connection will not be available for some time. In the interim, two alternative means of acquiring processed speech are being pursued. One involves direct input of digitized speech to TX-2 and the duplication of FDP analysis programs in TX-2. The other involves a cumbersome path from the FDP via magnetic tape and the IBM 360/67 to TX-2. Both courses are being followed: the first because it will be faster and more flexible in the interim, the second because it can assure compatibility when the direct connection becomes available.

In the case of direct input to TX-2, progress to date consists of an extension of the APEX time-sharing system to handle the analog-to-digital conversion hardware on TX-2. The required programs are now running, and TX-2 can handle up to 6.4 sec of telephone quality speech (9-bit samples at a 10-kHz rate). A fast Fourier transform (FFT) program is being developed utilizing the parallel subword arithmetic capability of TX-2 which promises to be fast enough to be useful.

In the case of the FDP, speech is digitized by analog-to-digital hardware on a Univac 1219 which serves as an IO processor for the FDP. For example, the FDP performs an FFT every 12 msec, sending the results back to the 1219. From there they are written onto a 7-track digital tape which is hand carried to the IBM 360/67 where the data are read off and transmitted via the ARPA network to TX-2. This latter step is necessary because TX-2 handles only 9-track tapes. Of course, a reformatted tape could be made on the 360, avoiding the network transmission, but we wanted to get some experience with the net. To date, a few utterances have been transmitted successfully, but, as indicated above in Sec. II-C, network operation has not yet become a smooth procedure, and the current capability is too slow and tedious to be useful.

When the new TX-2 NCP becomes available, it will become possible to evaluate the utility of this interim transmission scheme.

The data display and labeling programs are being developed to make use of the TSP console configuration (i.e., two direct-view storage scopes, tablet, and keyboard). A worker will get a spectrogram display of an utterance, together with functions of time like the overall amplitude or first moment of the spectrum. Using the tablet pen, he can zoom in on short stretches of the spectrogram, getting amplitude-vs-frequency displays on the second scope. He will indicate boundaries using the pen and key in phonetic symbols from the keyboard. An important question is whether the DVST can provide an adequate display for speech data. Results so far are encouraging.

The four illustrations of the utterance "he took a walk" in Figs. 2(a) through (d) show the current state of the spectrogram display on TX-2 using the Tektronix 611 storage scope. The data followed the path from the FDP described above. The functions below the spectrograms are the overall amplitude vs time and the first moment of the frequency spectrum vs time. The illustrations show the effects of changing the "gain," which is attenuated in Figs. 2(c) and (d). The other effect to observe is the result of normalizing sample magnitudes on a section basis, which has been done for both Figs. 2(b) and (d). It is already clear that the storage tube can produce a spectrogram of good visual quality. The four illustrations give a hint of the great flexibility with which various parameters of the display can be diversified.

V. TX-2 HARDWARE

A. LDX Printer/Plotter

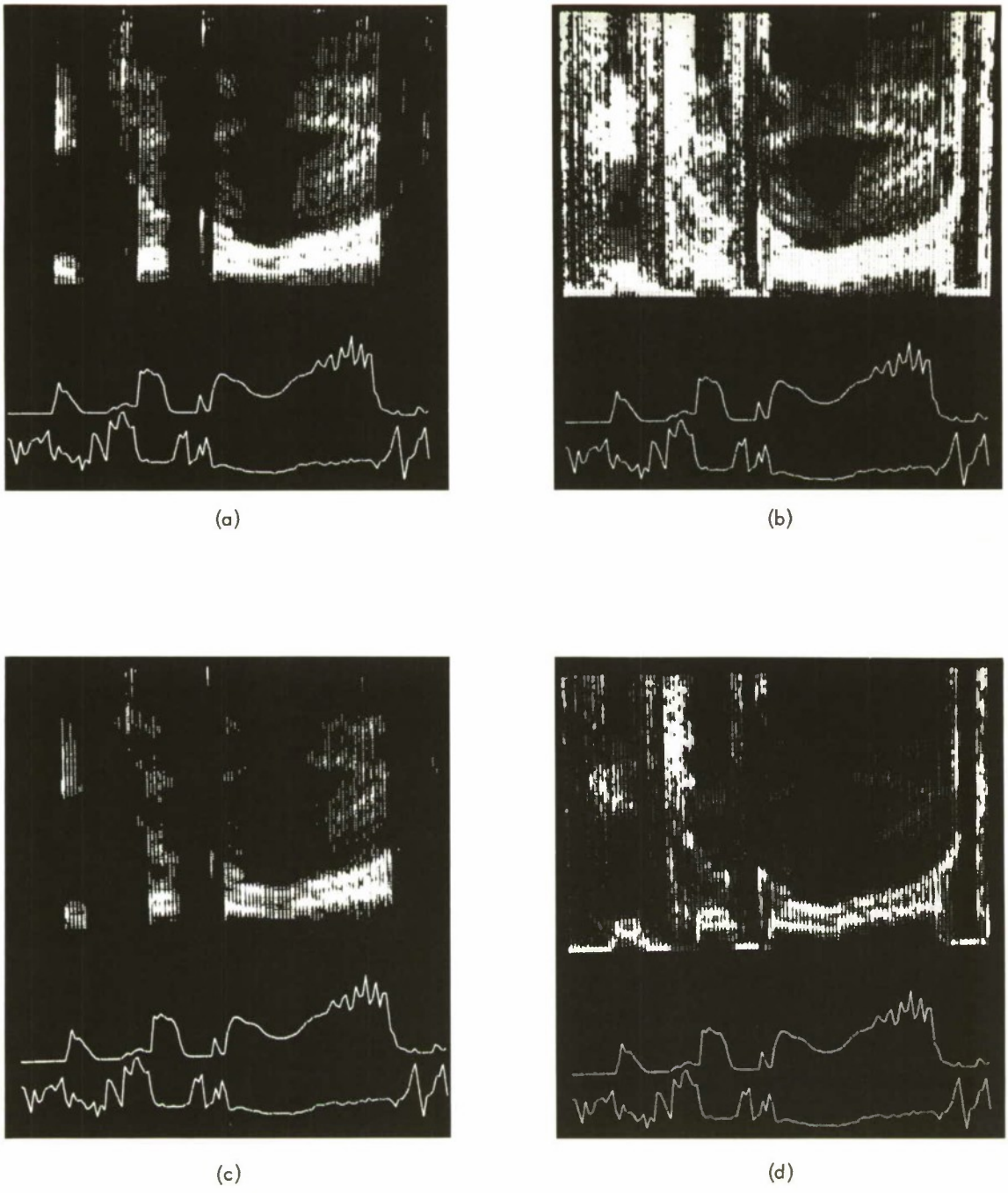
The LDX Printer/Plotter system uses a Digital Equipment Corporation PDP-8/L computer as a controller to drive a Xerox LDX Receiver terminal which produces hard-copy output by the Xerox process. Since the output is produced from a video raster, there is no hardware constraint on format. Printing formats (including character fonts and graphs) can be determined solely by software.

Hardware for the TX-2 to PDP-8/L interface and the PDP-8/L to LDX interface has been designed, constructed, and checked out. Hard copy has been produced from programs running in the PDP-8/L. Use of the LDX Printer/Plotter system on TX-2 is awaiting completion of the software for the PDP-8/L and modification of the TX-2 system software.

B. Hughes Scan Converter

A Hughes LCSC-1 scan converter was delivered to the Laboratory and connected briefly to TX-2 for evaluation. The scan converter, though currently more expensive than the direct-view storage scopes planned for the TSP system, is almost logically equivalent and should be superior for applications requiring brighter images, higher contrast, or black-on-white pictures.

The results of the preliminary evaluation were encouraging, and interface hardware to allow TX-2 to control the converter is now being developed. The new interface will allow exploration of the converter's selective-erase mode, a possible means of achieving something like the "write-thru" capability of the direct-view storage tube.



(a) (b)

(c) (d)

Fig. 2. "He taak a walk." Spectragrams an Tektranix 611 starage scape, shawing variatians in gain and narmalizatian. (a) and (b) have higher gain; (b) and (d) are narmalized.

DOCUMENT CONTROL DATA – R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Lincoln Laboratory, M.I.T.	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP None

3. REPORT TITLE

Graphics

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Semiannual Technical Summary – 1 December 1970 through 31 May 1971

5. AUTHOR(S) *(Last name, first name, initial)*

Forgie, James W.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
31 May 1971	28	2

8a. CONTRACT OR GRANT NO. F19628-70-C-0230 b. PROJECT NO. ARPA Order 691 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) Semiannual Technical Summary, 31 May 1971 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* ESD-TR-71-151

10. AVAILABILITY/LIMITATION NOTICES

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency, Department of Defense

13. ABSTRACT

The hardware for the Terminal Support Processor (TSP) system is in various stages of design, construction, and checkout. The two Digital Scientific Corporation Meta 4 microprocessors and main memory system have been delivered, but acceptance awaits the installation of revised memory controller and central processor boards. The secondary memory has been delivered to the interface vendor, and the interface has been checked out on a single processor system. A multichannel controller which will simplify the interfacing of the other peripheral equipment has been constructed and tested. A serial transmission and time multiplexing scheme for handling keyboards and console indicators has been designed. The design uses a standard TV composite video signal. Two minimal console stations have been built and tested. Error rates appear to be satisfactory for this application. TSP software effort has concentrated on developing the support facilities which run on TX-2 and handle assembly and compilation for the TSP. Code generators for a BCPL compiler for the BCOM machine, the processor for system software in the TSP, are being debugged and indicate good efficiency of compiled code. Diagnostics for the BCOM machine have been written and debugged on the Meta 4 simulator on TX-2 and have been tested satisfactorily in the Meta 4 system.

The communications hardware for connecting the Laboratory's IMP to the ARPA computer network was delivered and has been operating since January 1971. The IBM 360/67 Network Control Program (NCP) following the old official protocol was operating in February. TX-2 has made use of the network facility using interim network software. New NCP's for both machines are under development.

A structure definition facility is being added to the TX-2 BCPL compiler. Its installation in the front end of the TX-2 BCPL compiler will also make the features available to the BCOM BCPL compiler. A description of the structure definition facility is presented in some detail.

Work has started on a new program directed toward the development of speech understanding systems. A speech data base to support research in this area has been designed. Programs have been written to facilitate the transmisalon of processed speech from the Laboratory's Fast Digital Processor to TX-2 and for the display of speech spectrograms and time functions on the direct view storage scopes at TX-2.

Hardware for interfacing a Xerox LDX Receiver terminal to a PDP-8/L computer and the latter to TX-2 has been constructed and checked out. A Hughes LCSC-1 is being connected to TX-2 for evaluation as an alternative to the direct view storage tube. The results of a preliminary investigation were encouraging.

14. KEY WORDS

graphical communication
TX-2
time sharing
man-machine
display systems
programming languages